# A \* Search Algorithm

from collections import deque

class graph:

def \_\_init\_\_(self, adjac\_lis):

self.adjac\_lis = adjac\_lis

def get\_neighbours(self, v):

return self.adjac\_lis[v]

def h(self, n):

h = { 'A': 1, 'B': 1, 'C': 1, 'D': 1}

return h[n]

def a\_star(self, start, stop):

open\_lst = set([start])

closed\_lst = set([])

poo = {}

poo[start] = 0

par = {}

par[start] = start

while len(open\_lst)>0:

n = None

for v in open\_lst:

if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):

n=v; #Check

if n == None:

print('Path does not exist!')

return None

if n == stop:

reconst\_path = []

while par[n] != n:

reconst\_path.append(n)

n = par[n]

reconst\_path.append(start)

reconst\_path.reverse()

print('Path found: {}'.format(reconst\_path))

return reconst\_path

for (m,weight) in self.get\_neighbours(n):

if m not in open\_lst and m not in closed\_lst:

open\_lst.add(m)

par[m] = n

poo[m] = poo[n] + weight

else:

if poo[m]>poo[n]+weight:

poo[m] = poo[n]+weight

par[m] = n

if m in closed\_lst:

closed\_lst.remove(m)

open\_lst.add(m)

open\_lst.remove(n)

closed\_lst.add(n)

print('Path does not exist!')

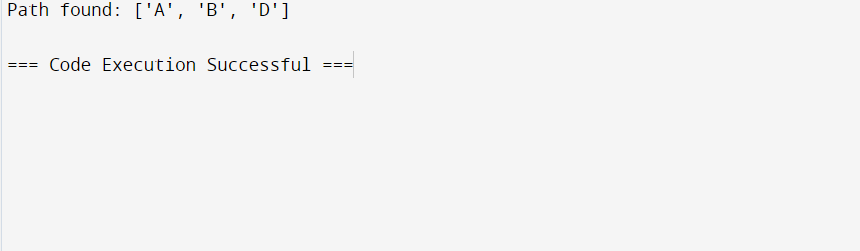
return None

adjac\_lis = { 'A': [('B',1),('C',3),('D',7)], 'B': [('D',5)], 'C':[('D',12)]}

graph1 = graph(adjac\_lis)

graph1.a\_star('A','D')

OUTPUT:

****